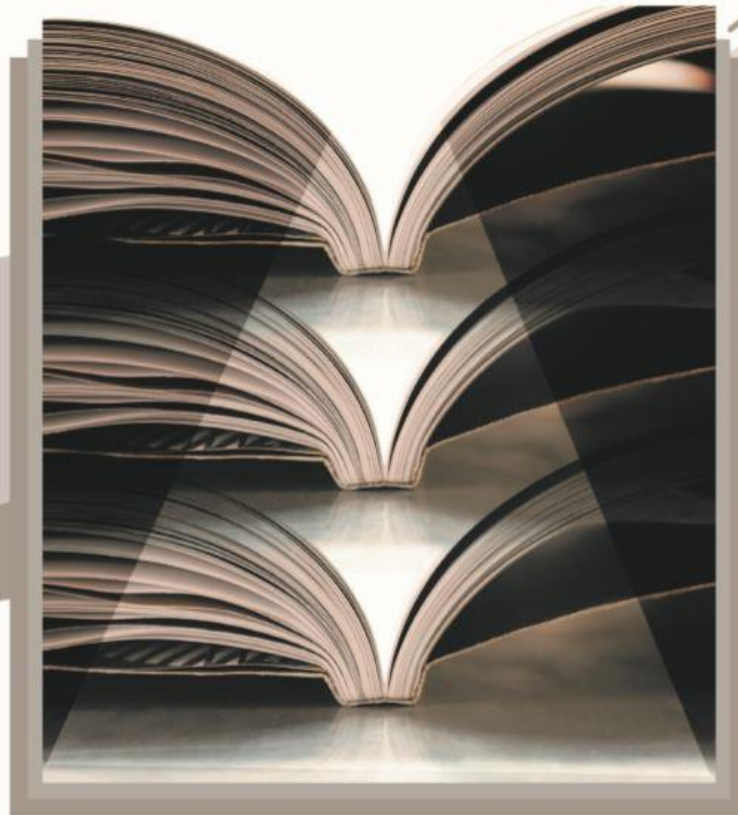


Liderazgo y producción de Cuerpos Académicos



Ma Teresa Prieto Quezada
José Claudio Carrillo Navarro
Lucila Patricia Cruz Covarrubias
Pedro Aguilar Pérez
Claudia G. Herrera

Primera edición, 2019

D.R. © 2019 UNIVERSIDAD DE GUADALAJARA
Centro Universitario de Ciencias Económico Administrativas
Perférico Norte 799
Núcleo Los Belenes
45100, Zapopan, Jalisco, México

D.R. © 2019 Fondo Editorial Universitario
Carrer La Murta 9-18
07820 San Antonio de Portmany
Ibiza, España

ISBN: 978-84-18080-62-3

Impreso en México
Printed in Mexico

Barrios LGBT. Caracterización de unidades económicas en la Zona Romántica de Puerto Vallarta, Jalisco, México	364
Andrés Enrique Reyes González, Jimena Odetti, Alberto Reyes González	
Impacto multidisciplinario de la globalización en la región Norte de Jalisco, México	370
José Manuel Núñez Olivera, Rodolfo Cabral Parra, Miguel Ángel Noriega García, Josefina Elizabeth Godínez Chavoya, Armando Antonio Domech González, Sandra Eva Lomeli Rodríguez	
Sector automotor, maquiladoras y articulación productiva en Cd-Juárez, Chihuahua, México - El Paso, Texas, EUA; intervención empresarial, consecuencias y desafíos	379
Juan José Huerta Mata, Ruth María Zubillaga Alva, Adriana Cordero Martín, César Omar Mora Pérez	
Observatorio de Educación Fiscal: La importancia del monitoreo en la opinión fiscal	394
Patricia Gutiérrez Moreno, Sandra Berenice Cabrera Reynoso, Alfonso Eduardo Dávalos Abad	
¿Es ética la evasión fiscal? Réplica a estudio empírico de opinión	405
Marco Antonio Daza Mercado, Javier Ramírez Chávez, Cristian Omar Alcantar López, Antonio Sánchez Sierra, Livier Padilla Barbosa, Francisco de Jesús Mata Gómez	
El entorno fiscal del derecho al agua como derecho humano	428
Laura Margarita Medina Celis, Rigoberto Reyes Altamirano, Ramiro Torres Torres	
Las organizaciones sociales como alternativa para disminuir la pobreza en México	444
Joaquín Sánchez Flores, Sergio Porras Zarate, José Enrique López Amezcua	
El actuar del primer respondiente en el nuevo sistema de justicia penal y su impacto en el proceso judicial	453
Marisol Luna García	
Circuito de regulación inteligente para detección de faltas en sistemas de eventos discretos: definición, funcionamiento y aplicación	460
Karen Hernández-Rueda, María Elena Meda-Campaña, Bernardo Haro-Martínez	
Diseño de nanomateriales multifuncionales con aplicaciones tecnológicas	470
María Luisa Ojeda M., Irinea Yáñez S., Víctor Rentería T., Javier Gálvez G., Miguel Ojeda M., Celso Velásquez O.	
Towards Verifying Product Architectures in a Software Product Line	476
Hector A. Duran-Limon, Karina Aguilar, Perla Velasco-Elizondo, Leonardo Soto Sumuano	
Viabilidad estructural en la reutilización de contenedores marítimos para construcción urbana	483
Iván Porras Zárate, José Manuel Salas Tafuya, Leticia Galindo González, Elizabeth Valenzuela González	
¿Qué deben conocer la población y los tomadores de decisiones sobre los pronósticos extendidos de lluvia y huracanes, el cambio climático, la contaminación atmosférica y las energías alternativas? . . .	490
Héctor Ulloa Godínez, Mario E. García Guadalupe, Omar García Concepción, Ángel Meulenert Peña, Ulises Ramírez Sánchez, Jaime Alcalá Gutiérrez	
Planes de mejora y calificaciones del Área de Comunicación: Proyectos de Fomento Lector	496
MLH. Miguel Ángel Galindo Núñez	

HECTOR A. DURAN-LIMON[♦], KARINA AGUILAR,
PERLA VELASCO-ELIZONDO, LEONARDO SOTO SUMUANO

Abstract

Software product line (SPL) engineering has proven to improve software quality and shorten costs and development time. Some research has been carried out to verify SPLs. However, most of this research has focused on verifying consistency of feature models. Moreover, approaches are still required for verifying structural aspects of product architectures with respect to the commonality and variability of an SPL architecture. Given that product line engineers often customize the product architecture by hand during the application engineering phase, the derivation and customization processes of the product architecture can be error-prone. In this paper, we present the Ontology-based Product Architecture Verification (OntoPAV) framework, which verifies structural aspects of product architectures. Our solution uses a language-independent model to specify the product line architecture. We use ontology formalism to capture the commonality and variability of an SPL architecture. Ontology reasoning engines are employed to check inconsistencies of product architectures. Furthermore, we use model-driven techniques to automate the ontology generation process. Our approach is illustrated with a voice over IP (VoIP) SPL case example.

Keywords: verification, *software* product lines, *software* architecture

Introduction

Software product lines (SPLs) have proven to improve software quality and shorten costs and development time (Krueger, 2006; Weiss, Clements, Kang, & Krueger, 2006). An SPL is a family of software systems that share a set of common assets (commonly referred to as the SPL commonality) but also have some other characteristics that make them different from each other (commonly named as the SPL variability). Feature modeling (Batory, 2005) is a technique that allows for specifying commonalities and variabilities in an SPL. A feature represents an increment in product functionality (Batory, 2005). Features are commonly organized as a tree, which is represented as a feature diagram. A product configuration involves a selection of features from an SPL. Such a configuration represents a member of the product family.

Software architecture (Bass, Clements, & Kazman, 1998) is a discipline that is emerging as a software design approach able to reach higher abstraction levels, hence, alleviating the software development complexity and making this development process less error-prone. Architecture description languages (ADLs) (Medvidovic & Taylor, 2000) represent formal notations for describing software architectures in terms of coarse-grained components and connectors.

Several efforts have focused on verifying consistency of feature models (D. Batory, 2005; Trinidad,

[♦] UdeG-CA-125, Tecnologías de la información y de la comunicación. ✉ hduran@cucea.udg.mx.

Benavides, Durán, Ruiz-Cortés, & Toro, 2008; Zhang, Zhao, & Mei, 2004; Yan, Zhang, Zhao, & Mei, 2009; Mendonça, Wasowski, & Czarnecki, 2009; Segura, 2008). Other research has focused on verifying consistency of the SPL architecture (Czarnecki & Pietroszek, 2006; Thaker, Batory, Kitchin, & Cook, 2007; Satyananda, Lee, & Kang, 2007; Kishi y Noda, 2006; Brito, Rubira, & de Lemos, 2009; Lopez-Herrejon & Egyed, 2010). However, to the best of our knowledge no approaches have been presented to verify compliance of structural aspects between the product architecture and the SPL architecture when the SPL language is an ADL such as Koala (van Ommering, van der Linden, Kramer, & Magee, 2000). Examples of such verifications include checking that all Mandatory elements defined in the SPL architecture are present in the product architecture as well as verifying that component interconnections of the product architecture follow the same sequence order defined by the SPL architecture.

In this paper, we present the Ontology-based Product Architecture Verification (OntoPAV) framework, which verifies that structural aspects of a product architecture conform to the SPL architecture. Our solution uses a language-independent approach to specify the product line architecture. We use ontology formalism to capture the commonality and variability of an SPL architecture. In particular, we employ Pellet (Sirin, Parsia, Grau, Kalyanpur, y Katz, 2007), an ontology reasoner, to check satisfiability (*i. e.*, consistency) of product architectures. Although our approach also involves the use of a formal method (*i. e.*, ontology formalism), the user does not require any knowledge about ontologies nor special training in any other formal method technique. We use model-driven engineering (MDE) techniques (Selic, 2006) to automate the ontology generation process. We have built a prototype to test our approach. Also, we developed an SPL of a voice over IP (VoIP) system as a case example.

The OntoPAV Framework

The overall approach of OntoPAV is depicted in Figure 1. The SPL architecture and product architecture are

transformed to the ontology OVerify by the Ontology Generation Engine. The ontology OVerify is used to verify structural aspects of the product architecture are compliant with the SPL architecture. In order to achieve language independence, a specific engine is generated for a specific SPL language. We use a similar approach based on our previous work (Duran-Limon, Garcia-Rios, Castillo-Barrera, y Capilla, 2015) to achieve language-independence, whereby an arbitrary SPL language (*i. e.*, an ADL) can be integrated into OntoPAV. For this purpose, we use the meta-metamodel MMO defined in (Duran-Limon *et al.*, 2015). With the aid of tool support, the metamodel Mo of the SPL language to be integrated is mapped to the meta-metamodel MMO. Based on this mapping, such a tool generates an instance of the Ontology Generation Engine which is specific to a particular SPL language.

The Verification Manager is in charge of coordinating the checks performed and also generates the verification result. The Verification Manager checks consistency of the ontology OVerify with the aid of the Ontology-based Reasoning Engine. In case one or more inconsistencies are detected, the Verification Manager asks the Debugger Engine to find the origin of such inconsistencies. In order to find an inconsistency, the Debugger Engine checks consistency on OVerify_{subset}, which represents different subsets of OVerify.

Verification Process

Our proposed Verification process encompasses the following three stages:

1. **Domain modeling.** The first stage encompasses the domain modeling activity and involves the following sub-tasks: (i) definition of the feature model, and (ii) definition of the SPL architecture containing the mappings between features and architecture elements. For these two sub-tasks we use the visual feature tree editor suggested in (Gomez & Ramos, 2010) and a complementary tool for editing the SPL architecture (Duran-Limon *et al.*, 2015).
2. **Product architecture modeling.** The second stage concerns the application modeling activity,

which involves the following: (i) the selection of the variants in the feature model, (ii) the definition of the product-specific architecture. Here, we use the tool defined in (Duran-Limon *et al.*, 2015) for editing the product architecture.

3. Product architecture verification. Finally, the third stage regards the verification of the defined product architecture using tool support.

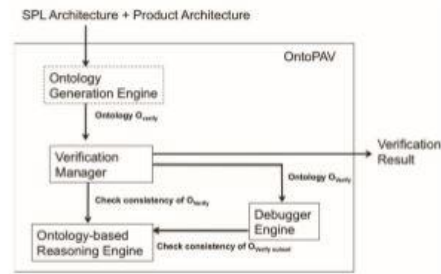
OntoPAV verifies the following structural aspects of product architectures: 1. Presence of elements. 2. Component Interconnections.

Verifying the presence of elements regards identifying whether the elements contained in the product architecture are valid or not as well as identifying whether there are missing elements, and they can be: Mandatory elements, elements with an Alternative relationship, elements with an OR relationship, presence of a variation point or a variant, and presence of any element not defined in the SPL architecture. The latter two are always invalid. For example, it is checked that all the components that belong to the commonality of the SPL architecture (*i.e.*, Mandatory elements) are present in the product architecture. It is also verified that an Alternative relationship in the SPL architecture reifies one and only one variant in the product architecture.

On the other hand, verifying component interconnections involves checking that components and connectors in the product architecture are interconnected in the right sequential order and with the right interface types.

Due to paper length restrictions, in this paper we only describe the Ontology Generation Engine and focus on describing our approach to verifying the presence of elements with Mandatory, Alternative, and OR relationships.

Figure 1
The Architecture of OntoPAV



Ontology Generation Engine

As said earlier, our approach is based on the work of Wang *et al.* (Wang, Li, Sun, Zhang, & Pan, 2007) where ontologies, represented in OWL (Horridge, Drummond, Jupp, Moulton, & Stevens, 2009), are used to model feature relationships. In our case, we use ontologies to model the architectural semantic relationships in SPL architectures and product architectures. OWL is based on description logic (DL) (Baader, Calvanese, McGuinness, Nardi, & Patel-Schneider, 2007), which is a family of logic-based knowledge representation formalisms. We use the Manchester OWL Syntax (Horridge *et al.*, 2009) to illustrate the OWL descriptions used for the verification. This syntax is mainly employed by many ontology editing tools such as Protégé (Horridge *et al.*, 2009). We use this syntax because it is easy to write and read. In addition, we use the Turtle syntax (W3C, Turtle, n.d.) to represent ontologies. The Turtle syntax is supported by most ontology-based reasoning engines. We use Turtle because its syntax is less verbose and more user-friendly than other formats. In an ontology, the domain is described in terms of individuals, classes, and object properties. Individuals are instances of classes and represent the basic units in the domain. Classes are sets of individuals with similar characteristics. Object properties represent binary relationships between individuals. Class constructors include the use of intersection, union, and complement operations as well as the existential and universal quantifiers, which in OWL are denoted

by the keywords *some* and *any*, respectively. Class hierarchies can be defined by using the subclass property. A subclass is a subset of individuals of its parent class. A necessary condition is represented as an anonymous Superclass whereas a necessary and sufficient condition is represented as an equivalent class in the Manchester OWL Syntax format. Both conditions are also called restrictions. These and other features of OWL can be used to give a precise and unambiguous meaning to the descriptions of the domain.

As mentioned earlier, we use model-driven engineering (MDE) techniques (Selic, 2006) to automate the ontology generation process. More concretely, the Ontology Generation Engine performs a model-to-text transformation whereby both the SPL architecture and the product architecture are transformed to an instance of the ontology OVerify, such a transformation is carried out as described below.

Transformation Rules. Phase 1

The following transformation rules are employed to transform the SPL architecture to the ontology OVerify¹:

Rule 1. Transform Mandatory elements to OWL classes and OWL restrictions. The Mandatory elements involve component and connector elements in the case of PL-Xelha. Therefore, in our running example component and connector elements of the SPL architecture are transformed to OWL classes that keep a subclass relationship with the OWL class Root representing the whole SPL architecture. For instance, the component *c src stub* is transformed to an OWL class named *c src stub*. In addition, for each Mandatory element the Ontology Generator defines an equivalence statement which is a necessary and sufficient condition that binds the class to an existential restriction. For example, the existential restriction *has C sink stub some c sink stub* is defined in an equivalence statement within the OWL class *c src stub*. Furthermore, an equivalence statement is defined within the OWL class Root with the existential restriction *has Root some Root*. Finally, for each Mandatory element a necessary condition is defined as a subclass statement involving an existential restriction e.g. *has Elementj some Elementj*, in the OWL class Root. The Mandatory

elements in our running example are *c src stub* and *c sink stub*, thus, the following restrictions were defined as two subclass statements in the class Root:

- *hasC src stub some c src stub hasC sink stub some c sink stub*

Rule 2. Transform Alternative elements to OWL classes and OWL restrictions. The Alternative elements involve variation points whose variants have an Alternative relationship. In our running example, Alternative elements include optional components and optional connectors, thus, both of them are transformed to OWL classes that keep a subclass relationship with the OWL class Root. The variants associated with a variation point are also transformed to OWL classes keeping a child relationship with its variation point. Similarly to rule 1, the Ontology Generator defines an equivalence statement with an existential restriction for each Alternative element and each variant element. Such a restriction is also placed as a subclass statement within the OWL class Root. Finally, for each Alternative element a necessary condition is defined as a subclass statement involving an existential restriction stating that one and only one variant can instantiate the variation point. This is achieved by defining a restriction, as a subclass statement in the class of the variation points, that specifies all the possible combinations of the variant that is instantiated and those variants that are not. For example, the variants associated with the variation point *vp stream conn* are *c many2many* and *c one2one*. Therefore, the Ontology Generator places a subclass statement within the OWL class *vp stream conn* with the following restriction:

- *not (hasC many2many some c many2many) and hasC one2one some c one2one or (not (hasC one2one some c one2one) and hasC many2many some c many2many)*

Rule 3. Transform OR elements to OWL classes and OWL restrictions. The OR elements involve variation points whose variants have an OR relationship. In our running example, OR elements include optional components and optional connectors, thus, both of them are transformed to OWL classes that keep a subclass relationship with the OWL class Root. For example, *vp*

english, vp spanish, and vp french, which conform an OR relationship are transformed, each one of them, to an OWL class. The variants associated with these variation points, namely c english, c spanish, and c french, are also transformed to OWL classes keeping a child relationship, each one with its corresponding variation point.

Similarly to the previous two rules, the Ontology Generator defines an equivalence statement with an existential restriction for each OR element and each variant element. Finally, a necessary condition is defined as a subclass statement involving an existential restriction stating that at least one variant must instantiate the variation point. This is performed by defining a restriction, as a subclass statement in the OWL class Root, that specifies that one or more variants must be instantiated. In the case of the example stated above, the Ontology Generator places a subclass statement within the OWL class Root with the following restriction:

- hasC english some c english or hasC spanish some c spanish or hasC french some c french

Transformation Rules. Phase 2

The following transformation rules are employed to transform the ontology OVerify¹ and the ontology OQuery to the ontology OVerify:

Rule 1. Transform the elements that are present and those that are not present in the product architecture to an OWL existential restriction. The elements in question are components and connectors of the product architecture. Such components and connectors in the SPL architecture may be either Mandatory elements or variants. The Ontology Generation Engine makes use of the OQuery to find the elements that are present and those that are not. We achieve the former by requesting those elements that are components and those that are connectors in the product architecture:

- ((isPcomponent some Pcomponent) or (isPconnector some Pconnector))

The elements that are not present in the product architecture are obtained by subtracting those ele-

ments that are in the product architecture to the set of elements that are in the SPL architecture as follows:

- ((isSPLcomponent some SPLcomponent) or (isSPLconnector some SPLconnector) or (isSPLvariant some SPLvariant)) and not ((isPcomponent some Pcomponent) or (isPconnector some Pconnector))

More concretely, this engine transforms the elements that are present in the SPL architecture but are not present in the product architecture as well as those elements that are present in both the SPL architecture and the product architecture to an existential restriction. This existential restriction is a necessary and sufficient condition that is defined as a equivalence statement in an OWL class named E Product.

In the case of our running example the Ontology Generator Engine produces the following restriction statement that specifies the elements that are present in the product architecture:

- hasC graphical linux some c graphical linux and hasC melp linux some c melp linux and hasC one2one some c one2one and hasC many2many some c many2many
- and hasC src stub some c src stub

We present below an excerpt of the statement produced that specifies the elements that are not present in the product architecture (the complete statement is presented in (Duran-Limon, 2017)):

- not (hasC recording some c recording) and not (hasC english some c english) and not (hasC spanish some c spanish) and not (hasC french some c french) and not (hasC sink stub some c sink stub) and not (hasC graphical macos some c graphical macos) and not (hasC graphical windows some c graphical windows) and not (hasC textual linux some c textual linux)
- and not (hasC textual macos some c textual macos) and not (hasC textual windows some c textual windows) and not (hasC melp macos some c melp macos)
- and not (hasC melp windows some c melp windows)

Note that we have split the restriction statement into two parts for clarity only as both of them form a single restriction. In phase 1 and phase 2, we used Acceleo (Foundation, 2018) to transform the models of the SPL architecture and product architecture to text in Turtle (W3C, Turtle, n.d.) representing the ontologies OVerify¹ and OQuery. In phase 3, we employed a program in Java to transform the latter ontologies to OVerify.

Conclusions

In this paper we have presented the OntoPAV framework, an approach to verify conformance of structural aspects between a product architecture and the SPL architecture. We rely on the use of ontologies to perform such a verification. Importantly, the user does not need to have skills on ontologies to use our approach, rather, we use model-driven techniques to automate the ontology generation process. We use our previous work to support language-independence with respect to the SPL language.

Future work regards elaborating on the details of how we verify that the component interconnections of the product architecture are compliant with the interconnections of the SPL architecture, as well as, providing details of how the Debugger operates. We will also perform an evaluation of the scalability of our solution.

Bibliography

- Baader, F.; Calvanese, D.; McGuinness, D. L.; Nardi, D. y F. Patel-Schneider (2007), *The Description Logic Handbook*, New York, NY, USA, Cambridge University Press.
- Bass, L.; Clements, P. y R. Kazman (1998), *Software Architecture in Practice*, Boston, MA, USA, Addison-Wesley Longman Publishing Co.
- Batory, D. (2005a), "Feature Models, Grammars, and Propositional Formulas", en *Proceedings of the International Software Product Line Conference (SPLC)*, 7-20.
- (2005b), "Feature Models, Grammars, and Propositional Formulas", en J. H. Obbink y K. Pohl (Ed.), *SPLC*, vol. 3714: 7-20, Springer.
- Brito, P. H. S.; Rubira, C. M. F. y R. de Lemos (2009), "Verifying Architectural Variabilities in Software Fault Tolerance Techniques", *WICSA/ECSA*, 231-240.
- Czarnecki, K. y K. Pietroszek (2006), "Verifying Feature-based Model Templates against Self-formedness ocl Constraints", en S. Jarzabek, D. C. Schmidt, y T. L. Velthuizen (Ed.), *GPCE*, 211-220, ACM.
- Duran-Limon, H. A. (2017), A Case Example for Illustrating Ontopav: Complementary. Disponible en: <http://hduran.cucea.udg.mx/publications/example1-paper-documents/verification.zip>.
- García-Ríos, C. A.; Castillo-Barrera, F. E. y R. Capilla (2015), *An Ontology-based Product Architecture Derivation Approach*, diciembre.
- IEEE Transactions on Software Engineering 41(12), 1153-1168. Foundation, T. E. (2018), Acceleo. Retrieved from <http://www.acceleo.org/>
- Gomez, A., y Ramos, I. (2010), Cardinality-Based Feature Modeling and Model-Driven Engineering: Fitting them Together. In *Vamos'10* (p. 61- 68), Horridge, M., Drummond, N., Jupp, S., Moulton, G., y Stevens, R.
- (2009), "A Practical Guide to building Owl Ontologies using the Protege-owl Plugin and Coode Tools Edition" 1.2, marzo. Retrieved from <http://www.co-ode.org/resources/tutorials/ProtegeowlTutorial.pdf>
- Kishi, T. y N. Noda (2006), "Formal Verification and Software Product Lines", *Commun. ACM*, 49(12): 73-77, diciembre.
- Krueger, C. W. (2006), "New Methods in Software Product Line Practice", *Commun. ACM*, 49: 37-40, diciembre.
- López-Herrejón, R. E. y A. Egyed (2010), "Detecting Inconsistencies in Multi-view Models with Variability", en T. Kuhn, B. Selic, M. P. Gervais y F. Terrier (Ed.), *ECMPA*, vol. 6138: 217-232, Springer.
- Medvidovic, N. y R. N. Taylor (2000), A Classification and Comparison Framework for Software Architecture Description Languages", *IEEE Transactions on Software Engineering*, 26: 70-93.
- Mendonça, M.; Wasowski, A. y K. Czarnecki (2009), "Sat-based Analysis of Feature Models is Easy", en *Proceedings of the 13th International Software Product Line Conference*, Pittsburgh, PA, USA, Carnegie Mellon University, 231-240.
- Ommering, R. van; Linden, F. van der; Kramer, J. y J. Magee (2000), The Koala Component Model for Consumer Electronics software", *Computer*, 33(3): 78-85.

- Satyananda, T. K.; Lee, D. y S. Kang (2007), "A Formal Approach to Verify Mapping Relation in a Software Product Line", en *Cit*, IEEE Computer Society, 934-939.
- Segura, S. (2008), "Automated Analysis of Feature Models using Atomic Sets", en *Software Product Lines, 12th International Conference, SPLC*, Limerck, Ireland, septiembre 8-12, 2008, proceedings, vol. 2 (workshops), 201-207.
- Selic, B. (2006), "Model-driven Development: Its Essence and Opportunities", en *Proceedings of the Ninth IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing*, Washington, DC, USA, IEEE Computer Society, 313-319.
- Sirin, E.; Parsia, B.; Grau, B. C.; Kalyanpur, A. y Y. Katz (2007), "Pellet: A Practical owl-dl Reasoner", *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2): 51-53 (Software Engineering and the Semantic Web).
- Thaker, S.; Batory, D.; Kitchin, D. y W. Cook (2007), "Safe Composition of Product Lines", en *Proceedings of the 6th International Conference on Generative Programming and Component Engineering*, New York, NY, USA, ACM, 95-104).
- Trinidad, P.; Benavides, D.; Durán, A.; Ruiz-Cortés, A. y M. Toro (2008), "Automated Error Analysis for the Agilization of Feature Modeling", *J. Syst. Softw.*, 81(6): 883-896, junio.
- W3c, turtle (n.d.), <http://www.w3.org/TeamSubmission/turtle/>. Consultado: 10 de octubre de 2017.
- Wang, H. H.; Li, Y. F.; Sun, J.; Zhang, H. y J. Pan (2007), "Verifying Feature Models using Owl", *Web Semant.*, 5: 117-129, junio.
- Weiss, D. M.; Clements, P. C.; Kang, K. y C. W. Krueger (2006), "Software Product Line Hall of Fame", *SPLC* (p. 237), IEEE Computer Society.
- Yan, H.; Zhang, W.; Zhao, H. y H. Mei (2009), "An Optimization Strategy to Feature Models' Verification by Eliminating Verification-irrelevant Features and Constraints", en S. H. Edwards y G. Kulczycki (Ed.), *Formal Foundations of Reuse and Domain Engineering: 11th International Conference on Software Reuse, ICSR 2009*, Falls Church, VA, USA, septiembre 27-30, 2009, Proceedings, 65-75, Berlin, Heidelberg, Springer Berlin Heidelberg.
- Zhang, W.; Zhao, H. y H. Mei (2004), "A Propositional Logic-based Method for Verification of Feature Models", en J. Davies, W. Schulte y M. Barnett (Ed.), *Formal Methods and Software Engineering: 6th International Conference on Formal Engineering Methods, ICFEM, 2004*, Seattle, WA, USA, noviembre 8-12, 2004, Proceedings, 115-130, Berlin, Heidelberg, Springer Berlin Heidelberg.